UniMem: Redesigning Disaggregated Memory within A Unified Local-Remote Memory Hierarchy

Yijie Zhong, Minqiang Zhou, and Zhirong Shen, *Xiamen University;*
Jiwu Shu, *Xiamen University, and Minjiang University*

https://www.usenix.org/conference/atc24/presentation/zhong

This paper is included in the Proceedings of the
2024 USENIX Annual Technical Conference.

July 10–12, 2024 • Santa Clara, CA, USA

978-1-939133-41-0

UniMem: Redesigning Disaggregated Memory within A Unified Local-Remote Memory Hierarchy

Yijie Zhong[1], *Minqiang Zhou*[1], *Zhirong Shen*[1],[*] *Jiwu Shu*[1,2]
[1]*Xiamen Key Laboratory of Intelligent Storage and Computing, Xiamen University*
[2]*Minjiang University*

Abstract

Disaggregated memory (DM) has been proposed as a feasible solution towards scaling memory capacity. A variety of memory disaggregation approaches have been introduced to facilitate the practical use of DM. The cache-coherent-based DM system, which relies on cache-coherent accelerator, can offer network-attached memory as NUMA memory. However, the current cache-coherent-based DM system introduces an extra address translation for each remote memory access. Meanwhile, the local cache mechanism of existing approaches overlooks the inherent issues of cache thrashing and pollution that arise from DM system.

This paper presents UniMem, a cache-coherent-based DM system that proposes a unified local-remote memory hierarchy to remove extra indirection layer on remote memory access path. To optimize local memory utilization, UniMem redesigns the local cache mechanism to prevent cache thrashing and pollution. Furthermore, UniMem puts forth a page migration mechanism that promotes frequently used pages from device-attached memory to host memory based not only on page hotness but also on hotness fragmentation. Compared to state-of-the-art systems, UniMem reduces the average memory access time by up to 76.4% and offers substantial improvement in terms of data amplification.

1 Introduction

Disaggregated memory has attracted significant interest due to its high memory capacity, resource efficiency, and capacity scalability [26, 45]. It separates computing and memory resources into computing nodes (CNs) and memory nodes (MNs), which are interconnected with high-speed network, such as Remote Direct Memory Access (RDMA) connections. Despite the advent of new memory technologies like CXL 1.0 memory devices and CXL 2.0 memory pooling, which can provide ample memory resources for a single server or rack-scale cluster [5], RDMA-based DM systems continue to play a pivotal role in supporting large-scale systems [26, 30, 37, 52]. Existing DM systems can be divided into three distinct categories. `Object-based DM systems` [34, 35, 41, 50, 54, 58] can provide fine-grain remote memory access through a key-value or a data-structure-based interface. They can achieve high performance by addressing the costly software overhead and restrictions of OS (e.g., page faults and data amplification). However, object-based DM systems require significant code modification of applications to utilize the new interfaces. `Page-based DM systems` [24, 26, 37] depend on virtual memory subsystem to expose remote memory transparently. These systems treat network-attached remote memory as a swap device and swap page between local page cache and remote memory in cases of a page fault. `Cache-coherent-based DM systems` [29–31, 52] leverage cache-coherent hardware to reap the advantages of both systems. Newly approached cache-coherent protocols, such as compute express link (CXL) [4] and CCIX [3], can interconnect processor with accelerator or co-processor as Non-Uniform Memory Access (NUMA) system. The cache-coherent accelerator is responsible for resolving memory access on device-attached memory. Relying on this, cache-coherent-based systems offer remote memory to CNs by exposing a range of `fake` physical memory space, which is perceived as the device-attached memory to the host of CNs. It enables the provision of transparent and fine-grained remote memory to CNs. In this remote memory mechanism, a CPU cache miss on fake memory range from the host is forwarded to the accelerator, which resolves the translation between the fake physical address and the remote memory address.

As the remote memory mechanism is realized by fake physical memory, each remote memory access introduces an address translation. This extra address translation introduces extra latency on remote memory access path. This could potentially overwhelm the accelerator, especially when there are multiple memory-intensive workloads running on powerful processors, generating hundreds of millions of CPU cache miss events per second. Furthermore, existing works organize local cache for remote memory on the device-attached

[*]Corresponding author: Zhirong Shen (shenzr@xmu.edu.cn)

memory of accelerator to minimize remote data fetching. Unfortunately, they overlook a fundamental aspect of the DM system: the cache size is invariably much smaller than the workload footprint. This discrepancy makes the local cache susceptible to thrashing and pollution [55, 61]. Therefore, it's crucial for the local cache mechanism to be resistant to thrashing and pollution.

In this paper, we explore the design of a high-performance cache-coherent-based DM system. We propose UniMem redesigning the remote memory mechanism to expose the remote memory pool directly to CN's physical memory space, thereby eliminating the indirection layer. We extend the memory hot-plug feature [11] of the OS to implement flexible remote memory management. Considering the local cache for remote memory on CNs has to withstand cache thrashing and pollution, we propose a local cache mechanism that reserves the majority of the cache space for frequently accessed pages and promptly evicts pages with little or no reuse. Moreover, we keep a record of remote address (address in remote memory pool) of evicted blocks to detect the reused pages [2]. Access to device-attached memory is slower than access to host memory. We propose a comprehensive fully-used page promotion scheme. It determines which page in device-attached memory should be migrated to the host memory based on both page hotness and hotness fragmentation. Therefore, workloads can exploit the benefit of faster host memory and bypass the overhead of cache-coherent interconnect. We facilitate the batch promotion of frequently used pages on a per-process basis to amortize page migration overhead. To demonstrate the effectiveness, we compare UniMem to Kona [30] and its variations on a set of typical workloads. UniMem reduces average memory access time by 33.4% and 24.1%, compared to Kona and its variation, respectively. In terms of data amplification, UniMem reduces it by 5.2–7.9×. We open source UniMem at `https://github.com/yijieZ/UniMem`.

The main contributions of this paper are:

- We conduct an in-depth analysis for existing DM systems, demonstrating the overhead from extra indirection layer and the inefficiency of local cache mechanism in the DM scenario (§2.2).
- We propose UniMem, a new design for a cache-coherent-based DM system. It eliminates the indirection layer by constructing a unified local-remote memory hierarchy. It also incorporates a thrashing-resistant local cache on device-attached memory and a comprehensive page promotion scheme that promotes pages based on page hotness and hotness fragmentation (§3).
- We conduct a comparative analysis of UniMem and the state-of-the-art cache-coherent-based DM systems across a wide variety of workloads, focusing on metrics such as average memory access time and data amplification. We evaluate the benefits offered by each design technique of UniMem. Furthermore, we evaluate the performance of UniMem under a variety of system configurations, including parameters like cache block size, host memory capacity and set associativity (§4).

2 Background and Motivation

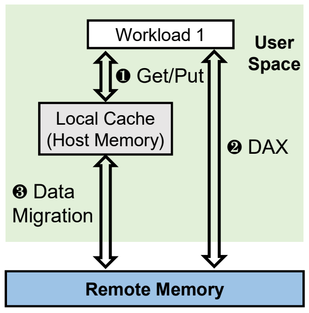
2.1 Disaggregated Memory

Basics of disaggregated memory: By separating computing and memory resources into different network-attached pools [24, 37, 41, 58], DM resolves the tight coupling of hardware resources in the traditional monolithic server model. Therefore, it can achieve high resource utilization and good elasticity. The compute pool has multiple CNs, each of which is a server that comprises powerful CPU cores and a small amount of memory. While the memory pool has many MNs, each MN is a server equipped with large DRAM capacity but limited computing power. The connection between compute and memory pools is typically enabled by microsecond-level RDMA network technique, which requires an RDMA NIC for each CN and MN.

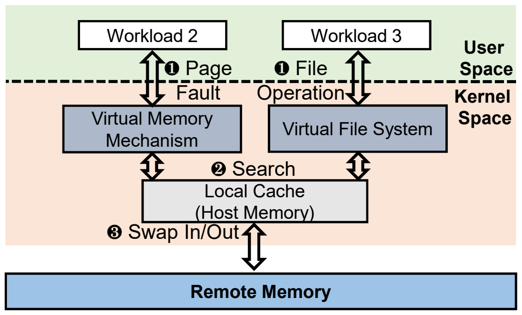
Taxonomy of disaggregated memory: Existing DM systems can be classified into the following branches: object-based, page-based, and cache-coherent-based DM systems.

`Object-based DM systems` expose a key-value or a data-structure-based interface for upper-layer applications to facilitate the fine-grained manipulation of remote data [34, 35, 41, 49, 50, 54]. Since the object-based DM systems provide remote memory abstraction in user level, they can bypass the expensive kernel path. While achieving good performance, the object-based DM systems need intrusive code changes that transform legacy applications to new interfaces, hence needing expensive engineering efforts and easily introducing additional program errors. Figure 1(a) depicts that the applications access remote data through key-value storage interfaces (`Get` and `Put`, Step ❶). The remote data can be fetched from local cache of object-based DM system or directly accessed from remote memory (Step ❷). Object-based DM systems also decide the data migration between local cache and remote memory (Step ❸).

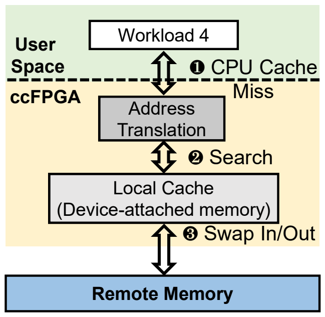
To improve compatibility, `page-based DM systems` rely on the traditional OS mechanisms and interfaces (e.g., virtual memory management [37, 38, 44] and virtual filesystem [24]), such that the upper-layer applications can leverage remote memory without needing any code modification. More specifically, it can map remote memory to the application's address space using virtual memory management and cache remote pages in the local cache. When applications attempt to access a page that does not reside in the local cache, the page-based DM system will trigger a page fault, which retrieves the requested page from the corresponding remote MN to the local cache over the network. Although having good portability, the page-based DM systems still suffer from performance degradation, caused by expensive kernel path (e.g., more than 60% of the throughput drops are caused by page faults and



(a) Object-based DM.



(b) Page-based DM.



(c) Cache-coherent-based DM.

Figure 1: Architectures of different DM systems.

TLB validations once moving only 25% of application's data remotely [30]). Figure 1(b) shows that applications access remote memory transparently by relying on virtual memory mechanism (Step ❶). It searches the requested data across the local cache at the host (Step ❷) and swaps in the data from remote memory if a cache miss occurs (Step ❸).

To ensure both transparency and good performance, `cache-coherent-based DM systems` propose to leverage cache-coherent protocol to track the `load` and `store` of applications at a fine granularity (e.g., cacheline), without relying on OS-level mechanisms nor requiring application modifications. Specifically, the emerging open industry interconnects based on the `PCI Express` (PCIe) interface (e.g., CXL [4] and CCIX [3]) realize the cache coherence between the CPU and the accelerator (e.g., FPGAs, GPUs, network/storage adapters, and customized ASICs). Kona [30] exposes a fake physical address space mapped in host physical memory space. The pages of this fake address space can be allocated to application as host local memory and always marked as present in application page table. When the CPU accesses the data from the fake memory space, it first searches on CPU cache as usual and turns to accelerator but not host memory while CPU cache miss happens. Cache-coherent accelerator will fetch the specific cacheline from either the local cache or the remote memory. Figure 1(c) shows that application accesses the fake memory range and misses in CPU cache (Step ❶). This CPU cache miss event is forwarded to cache-coherent FPGA (ccFPGA). After address translation (from fake physical address to remote memory address), Kona searches in local cache (Step ❷). While there is a miss in local cache, the corresponding data is swapped in from remote memory to satisfy the application access (Step ❸). Other cache-coherent-based DM systems [52] have the similar data flow with Kona [30].

The kind of cache-coherent-based disaggregated system which establishes rack-scale memory pool relying on cache-coherent interconnect with no networking interference [36] is not in the scope of this work, as the memory resources support memory semantics (i.e., `load` and `store`) natively. The cache-coherent-based DM system mentioned in this work represents works extending remote memory through network [29–31, 52].

2.2 Motivation

With the help of cache-coherent accelerator, cache-coherent-based DM system can provide high-performance remote memory transparently. However, there remain two fundamental limitations of existing systems.

Limitation#1 (Overhead from indirection layer). Cache-coherent-based DM system provides remote memory to applications transparently with the assistance of a cache-coherent accelerator. Since RDMA only supports operations similar to file operations (e.g., one-sided read and write), it treats remote memory pool as a swap device and encapsulates the details of data swapping (e.g., remote memory mapping and RDMA operations) within the ccFPGA.

Compared to the page-based DM system that simply needs to translate the virtual memory address to the physical memory address via page table for memory access, cache-coherent-based DM system introduces an additional step. It not only has to translate virtual memory address to fake physical memory address but also translate this fake physical memory address to remote memory address. This fake physical memory mechanism and address translation form an indirection layer. The indirection layer isolates CNs from each other and provides each CN with a private memory address space, as the remote memory pool is shared by all CNs. In particular, each time a CPU cache miss event occurs on fake physical memory range, the CPU forwards it to the accelerator over cache-coherent interconnect. The corresponding fake physical memory address is then transferred to the remote memory address via a hash table in Kona [30] or Remote Memory Management Unit in ThymesisFlow [52]. This remote memory address is used for searching local data cache on device-attached memory or fetching remote data from remote memory pool.

To investigate the impact of the indirection layer implemented in the Kona [30], Figure 2 illustrates the extra latency it introduces. We assume CXL [4] the cache-coherent interconnect that connects CPU and ccFPGA. The round-trip latency of a CPU cache miss event across CXL Link is 2×25 ns [5, 43, 48]. We then consider the address translation latency of indirection layer in the ccFPGA. The address

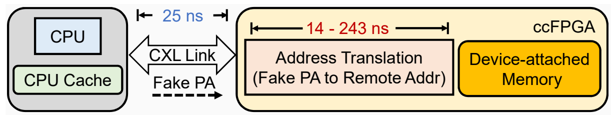


Figure 2: Indirection layer of Kona.

translation is realized by hash table lookup in Kona. Previous works [62, 63] have implemented high-performance hash tables on FPGA, showing that the search latency over the hash table ranges from 14 ns to 243 ns on state-of-the-art devices (i.e., Intel Stratix 10 FPGA [7] and Xilinx Alveo 250U FPGA [21]). The address translation of indirection layer introduces non-negligible latency to DM system.

We further conduct a simulated experiment to investigate the impact of indirection layer of Kona on the system performance. We use `Linux Perf` [9] to record the number of cache miss events of the CPU last-level cache and the elapsed time of running the applications on host memory. As every cache miss event in the CPU cache will undergo a round-trip on interconnect and introduce an additional address translation in the ccFPGA, the number of cache miss events is used to calculate the total runtime overhead of interconnect round-trip and address translation. The application runtime on Kona is simulated by adding application runtime on host memory to the interconnect round-trip and address translation runtime overhead. We assume the remote memory access is always satisfied by local cache on device-attached memory. We run eight real-world applications, which are abbreviated as `PR` (i.e., `Page Rank`), `GC` (i.e., `Graph Coloring`), `CC` (i.e., `Connected Components`), `RR` (i.e., `Redis-Rand`), `LR` (i.e., `Linear Regression`), `ETC` (i.e., `Facebook-ETC`), `YCSB-A` and `YCSB-B`. The details about experiment configurations can be found in §4.1. We set the latency of the address translation in the ccFPGA to 128 ns, which is the average latency reported in previous studies [62, 63].

Figure 3 shows that the address translation can occupy the total runtime from 18.7% (Linear Regression) to 58.7% (Graph Coloring), which is unacceptable. This additional address translation could be overwhelming for general-purpose accelerator, as there might be multiple powerful processors in a CN generating hundreds of millions of CPU cache misses under numerous memory-intensive workloads [29]. Otherwise, it consumes precious compute resources of the accelerator that could be used for boosting other processes such as local data cache searching. In multi-tenant environments [27, 32, 51], workloads would concurrently run on the same CN and compete for the CPU cache space. This competition could lead to higher CPU cache miss rate, resulting in increased address translation overhead.

Limitation#2 (Cache pollution and thrashing). DM systems use local memory (host memory or device-attached memory) as data buffer for remote memory to absorb remote access, thereby reducing network overhead. This makes local cache a crucial component for system performance. In page-based system, the virtual memory management organizes host memory as per-process page cache. It tracks the page hotness through 2Q (two LRU lists), namely inactive and active lists respectively [17], for page swapping between local and remote nodes. Cache-coherent-based system can establish data cache in device-attached memory. Kona [30] implements it as a 4-way set associative cache with the block size set to 4 KB, which is the common page size aiming to fully exploit spatial locality.

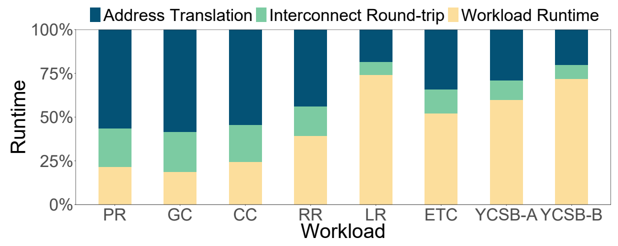
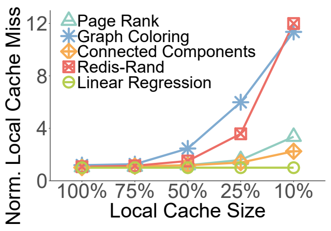


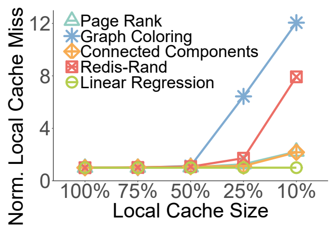
Figure 3: Performance breakdown (in percentage) of eight representative applications on Kona.

However, they overlook the vulnerability to cache pollution and thrashing issues that arise due to the limited capacity of the local cache. As proposed by previous work [59] and accelerator producers [6, 20], CNs are typically equipped with a small piece of memory around 10 GB, and the capacity of the memory pool might reach hundreds of gigabytes and even thousands of gigabytes (100s – 1000s GB) in the near future. Since the cache size is invariably much smaller than the workload footprint, memory-intensive workloads with a large working set size will compete for the limited local memory resources, when executed on the same CN. A previous study [61] on software cache observes that if the cache size is set to 10% of the working set size, around 72% of data in the cache is not reused before eviction. This is due to the fact that a smaller cache means a shorter observation window for the access pattern of workload. It lowers the likelihood of identifying locality to keep the frequently used (reused) data in the cache. Furthermore, the frequently used data might devolve into one-hit wonder (no request after insertion), polluting and thrashing the cache.

We conduct experiments on five typical workloads on both Kona 4-way set associative cache and the OS page cache to observe the trends of local cache miss rate at varying local cache sizes. We utilize `Intel Pin` [15] to trace the memory access operations for each application. The memory access sequences are replayed on simulated CPU cache for gathering the CPU cache miss events. Then, these cache miss events are replayed under different local cache mechanisms. The local cache size is progressively reduced from 100% to 10% of workloads' working set size. We evaluate the cache efficiency based on the rate of local cache misses normalized to the number of pages in the workload's working set. Figure 4 shows the normalized local cache miss rate for five workloads on Kona 4-way set associative cache and OS page cache,

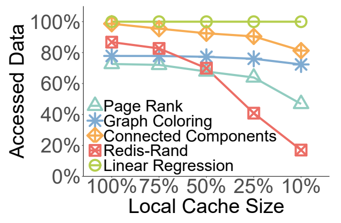


(a) 4-way set associative cache.

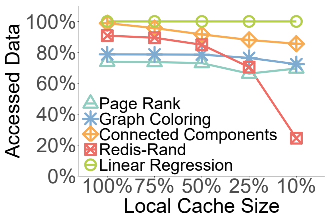


(b) Page Cache.

Figure 4: Normalized local cache miss rate.



(a) 4-way set associative cache.



(b) Page Cache.

Figure 5: The percentage of accessed data.

respectively.

Most of the workloads exhibit a similar trend that the normalized local cache miss rate increases as the local cache size diminishes on both local cache mechanisms. It increases steadily as the local cache size is reduced from 100% to 75% of the working set size on 4-way set associative cache and from 100% to 50% on the OS page cache. However, the normalized local cache miss rate increases rapidly when the local cache size is further reduced from 75% to 10% on 4-way set associative cache of Kona and from 50% to 10% on the OS page cache. This indicates that the efficiency of both Kona's 4-way set associative cache and the OS page cache is affected by the cache pollution and thrashing, as explained in prior works [55,61]. The local cache miss rate of Linear Regression shows no correlation with the local cache size, which can be attributed to its streaming access pattern that involves almost no data reused.

Furthermore, we discover that swapping remote data in 4 KB granularity leads to data amplification. We record the number of bytes accessed for every page that is swapped in, and gather the proportion of data accessed from that page when it is swapped out. Figure 5 shows the accessed data for five workloads on Kona 4-way set associative cache and OS page cache, respectively. We observe that the percentage of accessed data is influenced by the size of the local cache, with the exception of Linear Regression due to its access pattern. As the local cache size decreases, the accessed data percentage also declines. It declines modestly as the local cache size contracts from 100% to 75% on 4-way set associative cache, and from 100% to 50% on OS page cache. Then the accessed data percentage declines rapidly as the local cache size further shrinks from 75% to 10% on 4-way set associative cache, and from 50% to 10% on OS page cache. This can be attributed to the fact that pages have a shorter time to live in the cache as the local cache size decreases, resulting in them being swapped out before being reused. This data amplification is also a result of the coarse-grained data swap granularity, which is either constrained by the hardware (e.g., MMU and TLB) of the host processor in page-based systems, or by system design decisions in a cache-coherent-based systems.

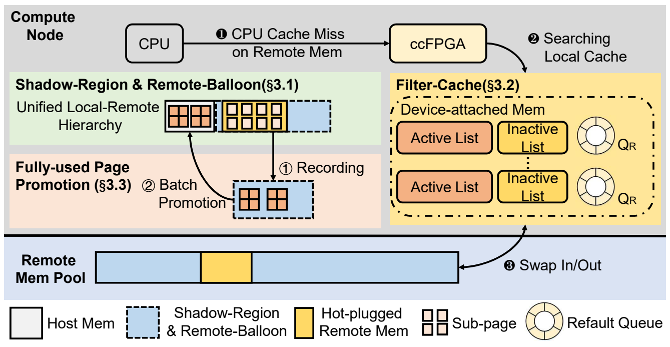


Figure 6: Architecture overview of UniMem.

3 UniMem Design

Motivated by the observations regarding the additional address translation of remote memory mapping and inefficiency of local cache in small capacity, we present UniMem, a high-performance cache-coherent-based DM system.

To eliminate the indirection layer overhead (additional address translation) on remote memory accessing path, UniMem exposes the remote memory pool directly to the physical memory space of CNs by Shadow-Region which relies on PCIe specification (§3.1). As all the CNs share the same remote memory address space, UniMem implements Remote-Balloon for remote memory synchronization with the help of Memory hot-(un)plug feature (§3.1). Figure 6 shows that the CPU cache miss on remote memory is forwarded to cache-coherent accelerator (Step ❶).

To satisfy the CPU cache miss from host on CNs, UniMem constructs Filter-Cache(§3.2) on device-attached memory which is a local cache for remote memory (Step ❷). It prevents local cache from pollution and thrashing by setting a small inactive list to filter out the one-hit wonders from active list, and reduces data amplification by using sub-page (512 B) as caching and swapping granularity (Step ❸).

Considering that the local memory in CNs includes host memory and device-attached memory, UniMem proposes fully-used page promotion scheme. It distinguishes and records fully-used pages in device-attached memory (Step ①). Then it promotes them to the faster host memory in batch (Step ②), thereby exploring local memory in a more refined manner (§3.3).

3.1 Shadow-Region and Remote-Balloon

Remote memory interconnected by RDMA serves as a swap device shared by CNs. The previous cache-coherent-based

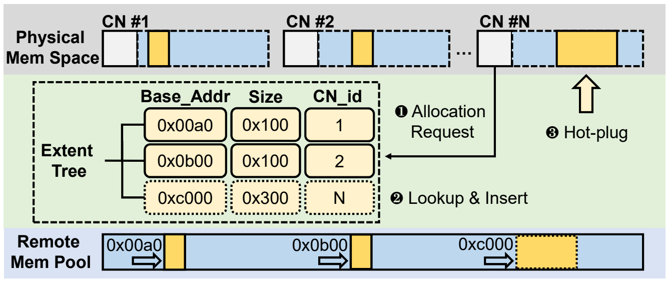


Figure 7: Shadow-Region and Remote-Balloon (§3.1).

disaggregated system [30, 52] encapsulates the complicated details of remote memory (e.g., RDMA semantic and remote memory mapping) inside cache-coherent hardware and exposes a fake physical memory range for host to utilize remote memory. However, the fake physical memory mechanism behaves like an indirection layer thereby introducing extra address translation. This turns out to be the bottleneck of critical path as described in §2.2.

In view of this, UniMem proposes to build a unified local-remote memory hierarchy that directly exposes the entire remote memory address range to eliminate the extra address translation. UniMem introduces Shadow-Region which presents the entire remote memory pool to each CN by mapping remote memory to system physical space of CNs. Thereby, CNs naturally treat remote memory as local physical memory resources. Subsequently, UniMem presents the resource management mechanism, Remote-Balloon, which is responsible for dynamic memory allocation, deallocation and synchronization between CNs. Once the CN requests memory resources from remote memory pool through Remote-Balloon, the specific remote memory range is hot-plugged for CNs. The OS of CN recognizes this hot-plugged physical memory and allocates it to applications straightforwardly.

Shadow-Region: UniMem assumes that each CN is connected with a cache-coherent accelerator over the PCIe physical layer. The PCIe standard defines a set of Base Address Registers (BARs) that devices can use to expose internal resources to specific host physical memory space range. Previous works [22, 39] leverage BARs to exploit the byte-accessibility of SSDs to extend host memory capacity with high-performance storage resources. When the host is powered on, the BIOS and OS check the BAR registers of PCIe endpoints and assign the specified physical memory region. In UniMem, we also use the PCIe BARs of cache-coherent accelerator to assign a physical memory space region for Shadow-Region. Consequently, the capacity of remote memory pool is required at the boot time of CNs. Figure 7 shows that Shadow-Region occupies physical memory space with the same size as remote memory pool in every CN. The pages belonging to Shadow-Region are distributed to applications on demand by page allocator of OS as normal physical memory resources. Similar to Kona [30], the memory request to Shadow-Region is redirected to the accelerator when it misses in CPU cache. But the physical memory address coming along with memory request can be used to address corresponding data in local cache or remote memory pool forthrightly without the necessity of extra address translation. This unified local-remote memory hierarchy design eliminates the extra address translation and reserves the compute resource of accelerator for other processes (local cache searching) in critical path.

In contrast to previous works [22, 39] on SSD, UniMem initializes Shadow-Region to be cacheable on CPU cache to leverage the benefit of it with the help of cache-coherent PCIe-based interconnect (e.g., CXL [4] and CCIX [3]). Additionally, Shadow-Region is not backed by real storage resources in accelerator but remote memory, while the device-attached memory acts as the local cache for remote memory and all the interactions between remote memory are concealed by UniMem.

Remote-Balloon: In UniMem, Shadow-Region is constructed in every CN. As a result, all CNs consider their exclusiveness of the whole remote memory pool which is shared in fact. The OS of CN considers remote memory as local physical memory and the page allocator can distribute these “exclusive” pages to applications as needed. However, it might cause fatal system errors without a proper synchronization mechanism. In Kona [30], a resource manager is responsible for allocating blocks of memory to CNs from the remote memory pool. It uses a hashmap to record the mapping between fake physical memory exposed to host and remote memory, which is the extra address translation described in §2.2.

UniMem proposes Remote-Balloon to implement dynamic and flexible memory management efficiently with the help of memory hot-(un)plug feature [11]. The memory hot-(un)plug feature supports physical memory onlining and offlining at runtime, which can be used to implement dynamic remote memory allocation and deallocation.

During the power-up process of CNs, the specific physical space of Shadow-Region is initialized to be offlined. At this time, remote memory is unavailable to CNs. Remote-Balloon introduces an extent tree to trace the allocated remote memory among CNs. Figure 7 shows that the entry in extent tree records the `Base_address` (base address on remote memory pool), `Size` and `CN_id` of each memory allocation. When a CN sends an allocation request (Step ❶), Remote-Balloon searches the extent tree to lookup an available remote memory range that satisfies the request and inserts an entry to the extent tree to reserve that remote memory range (Step ❷). Then, Remote-Balloon onlines the corresponding remote memory range within the Shadow-Region of the specific CN (Step ❸). After that, the allocated remote memory becomes visible to OS of specific CN as local physical memory resources to satisfy the memory requirement of applications.

The state of the allocated range is offlined on other CNs, so that other CNs cannot consume the same piece of remote memory, which avoids the conflicts between CNs. It also has a downside as it limits the sharing of remote memory pages

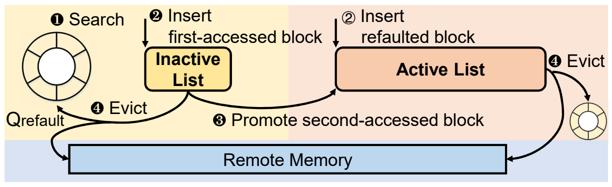


Figure 8: The illustration of Filter-Cache (§3.2).

among different CNs. Despite this drawback, it does not hinder the core objective of UniMem, which is to provide remote memory to CNs as exclusive local memory. It is efficient to minimize the metadata storage overhead with extent tree, as UniMem tries to allocate memory to the CN in sequential chunks as possible. Each time a CN can allocate remote memory in multiple times of memory-hotplug granularity. The memory hot-plug granularity depends on architecture, such as 128 MB in x86_64 and 16 MB in ppc64. This remote memory allocation mechanism is feasible to be implemented in a centralized or distributed approach.

With the help of memory-hotplug feature, it is convenient to release remote memory resources by unplugging memory block from CNs. When dynamically offlining a specific memory range, the OS migrates all pages off the affected memory block to another place. As the migration is finished, the state of corresponding memory range is changed to be offlined, and Remote-Balloon can reclaim this memory range for the next allocation. To accommodate the growth of remote memory pool, we recommend that the Shadow-Region should occupy physical memory range larger than remote memory pool at the CN's boot time. This setup ensures that UniMem can support the expansion of the remote memory pool as long as it remains within the capacity of the Shadow-Region. The Remote-Balloon is executed on host CPU with abundant generic compute resources but not in accelerator such as FPGA or SmartNIC. It is more proper to run the control path in host CPU, but these heterogeneous processors and the precious compute power of accelerator is reserved.

3.2 Filter-Cache

The DM systems leverage local memory resources of CNs to cache remote data for reducing remote data fetching. However, the restricted local memory capacity poses a challenge in designing an efficient local cache for memory-intensive workloads. It becomes difficult to identify frequently accessed (hot) data with limited cache space and prevent cache pollution and thrashing, as abovementioned observation in §2.2.

In view of this, UniMem introduces Filter-Cache. It partitions cache space into `active list` and `inactive list`. The active list is used to reserve popular data blocks to absorb as much remote data access as possible. The inactive list serves as a guard, containing the first-accessed block to protect the active list from thrashing and pollution. Filter-Cache maintains the history records of evicted blocks from active and inactive lists in a ring buffer, called `refault queue`, to extend the capacity of lists logically for recognizing popular data. The cache block size in Filter-Cache is set to sub-page for mitigating the data amplification. Furthermore, Filter-Cache divides the cache space into equal parts to construct multiple pairs of active and inactive lists which is similar to set associative cache for enabling parallel cache operations.

Figure 8 illustrates the data flow of Filter-Cache. When the block is fetched from remote memory pool, Filter-Cache first searches the refault queue to verify whether the block is previously evicted from active or inactive list (Step ❶). The block that is accessed for the first time is inserted at the head of the inactive list (Step ❷). If the block is refaulted (recorded in the refault queue), it is inserted at the head of the active list (Step ②). When the block in inactive list is accessed, it is promoted to the head of active list as popular data (Step ❸). When the block is evicted from inactive list or active list, the block is written back to remote memory if it is modified (Step ❹). The remote address of evicted block is logged in refault queue.

Splitting local cache into two lists prompts an immediate policy decision: what should be the size of each list? Linux's page cache roughly regulates that the active list does not exceed the inactive list [2]. However, this strategy can be inefficient as half of the cache space might be used for caching one-hit wonders. UniMem regulates that the size of active list can grow up to 90% of the cache space and the inactive list occupies at least 10% of the cache. When Filter-Cache starts in cold, most of the blocks flow into inactive list. At this time, the inactive list can even grow to occupy the whole cache. The second-accessed blocks in inactive list are considered as popular blocks and promoted to active list. The active and inactive lists are organized as LRU and FIFO lists, respectively.

As the active list grows and takes up more cache space, the size of inactive list diminishes, thereby reducing the lifespan of blocks on the inactive list. This implies that the observation window for the workload access pattern is shortened. It might be detrimental to cache efficiency, as no additional block is defined as popular to be promoted from the inactive list to active list. When the hot spot shifts, the blocks in active list become cold but still take up the cache space. As a result, the entire cache is halted. To address this issue, we draw inspiration from refault distance of Linux kernel [2] and ghost list of S3-FIFO [61]. UniMem adopts a refault queue to log the remote address of evicted blocks from both active and inactive lists, as these blocks are still logically buffered in the local cache. It provides a larger observation window for local cache mechanism to recognize popular blocks. The maximum number of entries in the refault queue is set to match the number of maximum caching blocks. When a block is evicted from active list, it is directly discarded and not inserted into inactive list. We regard the block evicted from active list as unvalued because the hot spot of workload has shifted or the block has become cold.

The swapping and caching granularity in Filter-Cache can be set up to any size. According to prior work [30], most of the contiguous accessed data in a 4 KB page spans a length of 1 to 4 cachelines (64 bytes). The remaining data may not be accessed before the page is evicted. In UniMem, the swapping and caching granularity is set to 512 bytes for lower data amplification. However, the finer granularity might incur higher software overhead when managing cache blocks (block promotion or eviction). It could also cause a rise in local cache misses and initiate more RDMA operations, as a coarser block essentially serves as a prefetch for data. Thus, Filter-Cache promotes or evicts blocks in batch for alleviating the software overhead and reduces RDMA operations by batching remote data fetching through optimization approaches, such as Doorbell Batch [42] and Scatter/Gather List [47]. We conduct experiments in §4.5 to discuss the benefits and drawbacks of fine-grained caching block.

It is complicated and inefficient to implement Filter-Cache upon FPGA. We propose to utilize the wimpy CPU cores in the accelerator [6, 20] for its control plane, thereby fully leveraging the various computation resources in the accelerator.

3.3 Fully-used Page Promotion Scheme

UniMem eliminates the extra address translation in remote memory access path and constructs a highly efficient local cache on device-attached memory for remote data. However, it still exhibits higher latency when accessing device-attached memory compared to host memory in CNs. It is critical to thoroughly explore the limited fast host memory for optimal system efficiency.

Unlike cache-coherent accelerator, which can flexibly determine the cache block size, the host is constrained to buffer data in page (4 KB commonly) due to hardware restrictions (e.g., MMU and TLB). The mismatch in cache block size between device-attached memory and host memory complicates the promotion scheme. Simply promoting data in page granularity from device-attached memory to host memory in terms of page hotness might be sub-optimal. As shown in previous works [28, 30] and discussed in §2.2, data amplification in page granularity could consume a significant amount of cache space for unnecessary data, severely impacting the efficiency of the data cache. This phenomenon is known as hotness fragmentation in prior work [28], which is caused by different accessing frequencies of sub-pages within a page.

UniMem proposes fully-used page promotion scheme. A fully-used page is defined as a frequently used page with low hotness fragmentation. It is considered more appropriate to promote these pages to host memory in order to reduce the overhead of cache-coherent interconnect. This is because a page with the same level of hotness but higher fragmentation is more likely to be cached by the CPU cache.

The fully-used page promotion scheme implements per-process promotion. Since the pages of remote memory are directly mapped into process address space as host memory in UniMem, they are tracked by per-process page cache [17, 19] (the same one in page-based DM system). UniMem selects the hot page depending on the process page cache and then evaluates the hotness fragmentation of them. UniMem measures the hotness fragmentation of a page by the state of each sub-page within the same page. As sub-pages of a page might be scattered across different lists on device-attached memory, UniMem assigns a `hotness score` to each sub-page based on the list they reside in. The sub-page in active list is given the highest score, while the sub-page in refault queue receives the lowest score. The variance of sub-pages' hotness scores represents the hotness fragmentation of a page. After collecting a set of promotion candidates, a batch promotion is activated. The batch promotion can amortize the software overhead of page migration, such as page table modification and TLB invalidations and shootdowns [18, 60].

4 Evaluation

We conduct comprehensive experiments to evaluate the performance of UniMem through simulation. We compare the effectiveness of UniMem to the state-of-the-art cache-coherent-based DM system under various local cache capacity (§4.2, §4.3 and §4.4). Next, we adjust the system configurations of UniMem to understand their influence on system performance, which includes cache block size (§4.5), host memory capacity (§4.6), and set associativity (§4.7).

4.1 Experiment Setup

Comparison systems: The cache-coherent-based DM system is comprised of two key mechanisms: the remote memory mechanism and the local cache mechanism. The remote memory mechanism can be of two types, including fake physical memory from Kona [30] and Shadow-Region and Remote-Balloon (SR&RB) from UniMem. The local cache mechanism is adaptable, as it can incorporate any software cache mechanisms from other systems, such as 4-way set associative cache of Kona, OS page cache [17] and Filter-Cache of UniMem. Therefore, we compare UniMem against the state-of-the-art cache-coherent-based DM system, Kona, and its variation.

- **Kona [30]:** Kona is the state-of-the-art cache-coherent-based network-attached DM system that proposes the use of cache-coherent accelerator for disaggregated memory implementation. It innovatively provides transparent access to remote memory for CNs via a fake physical memory mechanism and establishes a 4-way set associative cache for remote memory within the device-attached memory. The cache block size is set to 4 KB in Kona.
- **Kona-PC:** This is a variation of Kona, which substitutes the local cache mechanism with the OS page cache [17]. The OS page cache keeps track of the pages in cache using two LRU lists, namely the active and inactive lists. We regulate the size of these lists as Linux kernel that the active list is restricted from exceeding the size of the inactive list [2]. The cache block size is also set to 4 KB.

Table 1: Simulation configurations.

Memory Tier	Hardware Configuration	Average Latency
CPU Cache	L1D, 12-way, 48 KB per core L2, 20-way, 1.2 MB per core L3, 12-way, 24 MB shared [8]	20 ns (64 B)
Host Memory	DDR4 DIMMs [43,48]	80 ns (64 B)
Device-attached Memory	Intel's Sapphire Rapids with Intel FPGA implementing CXL at x16 width PCIe 5.0 interconnect [56]	150 ns (64 B)
Remote Memory	One-sided RDMA Operation through Mellanox ConnectX-3 [42,47]	2 µs (512 B) 4 µs (4 KB)

UniMem configuration: Without otherwise specified, we select the following configurations for the evaluation. We regulate the Filter-Cache (§3.2) that the active list can grow up to 90% of the cache space and the inactive list occupies at least 10% of cache space. The limit of records in refault queue is set to match the maximum number of cache blocks in cache. The cache space for refault queue is pre-allocated. As cache block size of UniMem is set to 512 bytes, the refault queue takes up approximately 1.5% of cache capacity. The number of pairs of active and inactive lists is set to 8. When the fully-used page promotion scheme (§3.3) is enabled, we assume 30% of local memory as host memory for page cache and 70% as device-attached memory for Filter-Cache. UniMem promotes 512 base pages (4 KB) in batch from device-attached memory to host memory.

Besides, we break down UniMem to demonstrate the effectiveness gained by each design technique as follows: (i) Shadow-Region and Remote-Balloon with 4-way set associative cache (SR&RB-4SC), which replaces Kona's remote memory mechanism (fake physical memory) with Shadow-Region and Remote-Balloon (§3.1), while maintaining the local cache mechanism as 4-way set associative cache; (ii) UniMem-NoPromote, which disables the fully-used page promotion (§3.3) of UniMem.

Workloads: We evaluate UniMem using three categories of memory-intensive workloads: in-memory key-value storage, graph analytics workload, and in-memory MapReduce workload. For the in-memory key-value storage system, we use Redis [16] and initiate a uniformly random key-value storage workload by Memtier [12] benchmark, referred to as `Redis-Rand`. We also execute Yahoo Cloud Serving Benchmark (YCSB) [33] workloads (`YCSB-A` and `YCSB-B`) on Redis, representing typical cloud services. In addition, we run the Facebook's ETC [27] workload on Memcached [10] by Mutilate [14], denoted as `Facebook-ETC`. For the graph analytics workload, we launch `Page Rank` using GraphLab [46] with the Twitter dataset [1]. Metis [40] is the in-memory MapReduce framework that we use to run the `Linear Regression`. The working set size of these workloads ranges from 4 GB (Redis-Rand) to 40 GB (Linear Regression).

Simulated configuration: We compare UniMem with other systems using Pin-based simulation. We gather the memory access operations, memory address and access data size of the workload using Intel Pin [15] and replay these operations on a simulated CPU cache to obtain the CPU cache miss events of the workload. The CPU cache configurations follow the real hardware specifications of Intel Xeon Silver 4314 processors [8]. Since each CPU cache miss comes to the local cache, we replay the CPU cache miss events in different local cache mechanisms to collect local cache miss events. The local cache miss is addressed by fetching remote data from the remote memory pool via RDMA operations (one-sided read). Based on the cache miss counts on different memory tiers, we can collect various performance statistics (e.g., average memory access time and remote fetching data size) of workloads under different DM systems.

Table 1 summarizes the simulation configurations. As the CPU cache configurations are identical to all workloads and systems, we disregard the latency differences between CPU cache levels and set CPU cache hit latency of all levels at an average of 20 ns, as in previous works [5, 48]. We assume CXL [4] as the interconnect between host processor and accelerator. The latency for a processor to load a cacheline from device-attached memory (end-to-end overhead for CXL reads) across CXL ranges from 150 ns to 175 ns [5, 43, 48] in Intel's Sapphire Rapids CPU with x16 width PCIe 5.0 interconnect. Therefore, we consider the hit latency of local cache on device-attached memory as 150 ns. According to prior works [47,57], we consider that fetching a 512 B or 4 KB block from remote memory to device-attached memory via RDMA introduces a latency of 2 µs or 4 µs, respectively. For the page promoted to host memory, we assume the host memory access latency as 80 ns. As described in previous works [13, 60], the overhead for promoting 512 base pages in batch is 2 ms. Specifically, Kona [30] introduces a hash table for extra address translation in the path of fetching a cacheline from device-attached memory to CPU cache. According to previous works [62, 63], the searching latency of hash table varies from 14 ns to 243 ns on the state-of-the-art devices (i.e., Intel Stratix 10 FPGA [7] and Xilinx Alveo 250U FPGA [21]). We consider it adds 14 ns of latency to the path in our experiments.

4.2 Average Memory Access Time

In this section, we first evaluate the average memory access time (AMAT) of six representative workloads on different DM systems. AMAT is considered a crucial performance indicator of DM system. We adjust the local cache size from 100% to 10% of workloads' working set size. As the CPU cache miss event of workloads is consistent across all systems, the presented AMAT is simulated without including CPU cache hit access to focus on the efficiency of DM system.

Figure 9 shows that UniMem surpasses both Kona and Kona-PC by demonstrating lower AMAT for most workloads across different local cache size configurations, with the exception

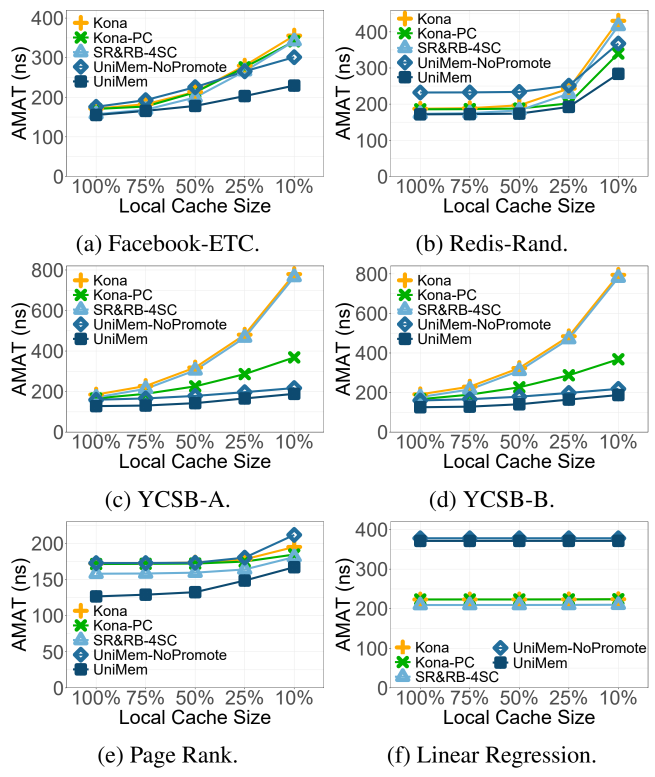


Figure 9: Average memory access time.

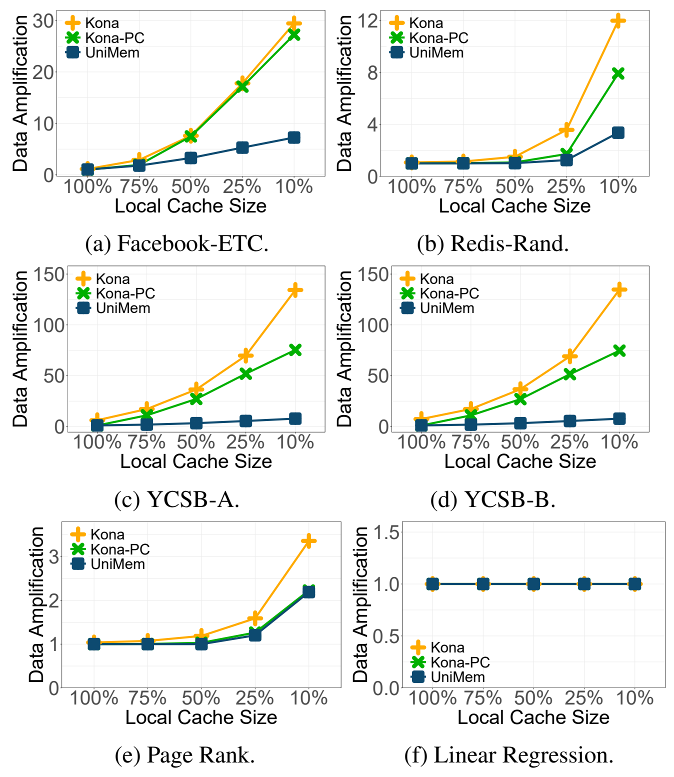


Figure 10: Data amplification.

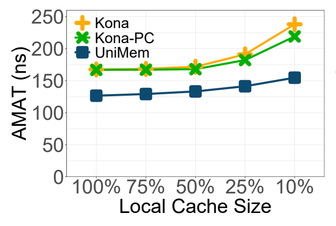
of Linear Regression. On average, UniMem achieves a lower AMAT by 33.4% and 24.1% compared to Kona and Kona-PC, respectively. As the local cache capacity decreases, the AMAT of all systems increases. However, UniMem exhibits a more gradual increasing trend and performs up to 76.4% and 49.1% (YCSB-B) better than Kona and Kona-PC in the scenario with limited cache capacity (10% of working set size), due to its local cache mechanism.

We break down UniMem to understand the benefits derived from each design technique. SR&RB-4SC shows a lower AMAT across all workloads compared to Kona with an average improvement of 7.1%. This is credited to the remote memory mechanism of UniMem, which eliminates the extra address translation overhead in Kona. The AMAT of UniMem-NoPromote varies according to workloads and local cache capacity configurations. UniMem-NoPromote presents lower AMAT compared to Kona and Kona-PC by 43.6% and 43.5% on YCSB workloads. It outperforms them in the low cache capacity scenario (10% of working set size) on Redis-Rand and Facebook-ETC. UniMem-NoPromote shows a higher AMAT than Kona and Kona-PC on Page Rank and Linear Regression. This is due to the Filter-Cache of UniMem-NoPromote adopting a finer cache block size (512 B) than 4-way set associative cache of Kona and OS page cache (4 KB), leading to more local cache miss events. It negates the benefit of reduced data amplification brought by finer cache granularity. The coarse cache granularity acts as a prefetch for subsequent data and reduces local cache misses, especially in workloads with a sequential access pattern or the scenario with sufficient cache space. UniMem outperforms UniMem-NoPromote by an average of 17.6% due to the fully-used page promotion scheme, which promotes frequently used pages to host memory reducing the overhead from cache-coherent interconnect.

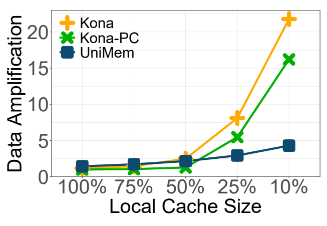
4.3 Data Amplification

To get further on local cache efficiency, we evaluate UniMem against other systems based on the amount of data fetched from remote memory pool. Retrieving a smaller amount of data from remote memory signifies a more efficient local cache mechanism. This suggests a reduction in data amplification and less network bandwidth being wasted.

We track the data fetched from remote by the local cache miss event across six representative workloads. Figure 10 illustrates the data amplification which is the amount of remotely fetched data normalized to working set size of workload with different local cache size configurations. It shows that a reduction in the local cache size leads to an increase in data amplification for most workloads across all systems. The data amplification of UniMem is minimally affected by the reduction in local cache capacity, maintaining an average of 2.6 times the working set size. In contrast, Kona and Kona-PC present an average of 20.6 and 13.6 times the working set size, respectively. The underlying reasons are two-fold. On the one hand, the local cache mechanism of UniMem presents higher cache efficiency than other systems, particularly in the scenario with low cache space. The small inactive list of

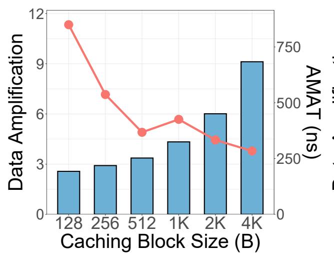


(a) AMAT.

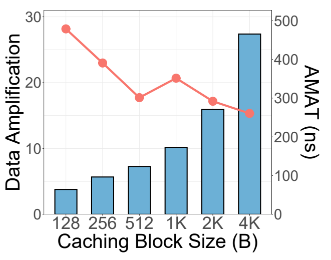


(b) Data Amplification.

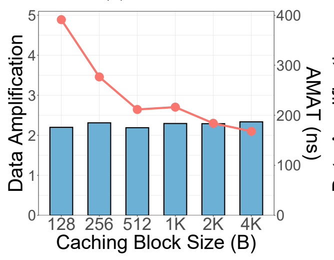
Figure 11: AMAT and data amplification with concurrent execution of multiple workloads.



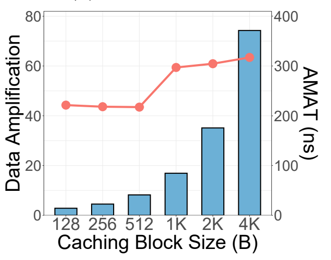
(a) Redis Rand.



(b) Facebook-ETC.



(c) Page Rank.



(d) YCSB-A.

Figure 12: Impact of cache block size. The bars illustrate data amplification, while the line indicates the AMAT of systems.

Filter-Cache prevents local cache thrashing, and the active list occupies the majority of the cache space for maintaining frequently used blocks. It leads to quick eviction of one-hit wonders and fewer refaulted swap-in of hot blocks. On the other hand, UniMem adopts a finer cache granularity than other systems. It means that each time a local cache miss occurs, UniMem fetches less data from remote memory pool. For workloads with a random access pattern or the scenario with limited cache space, it comes to lower data amplification and prevents local cache thrashing and pollution. Kona presents higher remote data fetching than Kona-PC and UniMem due to the conflicts on cache set.

4.4 Mixed Workload

We further evaluate the performance benefit of UniMem under several memory-intensive workloads, which simulates the concurrent execution of multiple workloads. It aims to demonstrate the system performance of DM systems in a more realistic scenario. The memory access operations from different workloads will compete for local cache, as local cache on cache-coherent-based accelerator cannot receive the hint of process from host.

The mixed workload includes four workloads: Redis-Rand, Facebook-ETC, Page Rank and YCSB-A. We simulate it by interleaving their memory access operations. The local cache capacity is adjusted from 100% to 10% of workloads' working set size. Figure 11 shows that UniMem presents the lowest AMAT compared to Kona and Kona-PC, achieving an average improvement of 26.9% and 24.1% respectively. For the data amplification, UniMem achieves the lowest data amplification in the scenario with limited local cache capacity (25% and 10% of working set size). It presents 2.9 and 4.1 times lower data amplification compared to Kona and Kona-PC, respectively. On both metrics, UniMem presents the least impact from reduction of local cache capacity.

4.5 Cache Block Size

We evaluate the performance of UniMem with different cache block sizes. It aims to demonstrate how the cache block size impacts the system performance. The cache block size affects both the local cache miss rate and remote data fetching size, which are essential metrics related to the AMAT and data amplification.

We evaluate the data amplification and AMAT of UniMem in the scenario with limited local cache capacity (10% of working set size) with cache block size ranging from 128 B to 4 KB. The latency for remote fetching of 128 B and 256 B blocks are set at 2 µs as 512 B block. For a 2 KB block, the remote fetching latency is set at 3 µs. We disable the page promotion scheme to avoid interference. Figure 12 shows that the data amplification increases as the cache block size grows across four workloads. The most significant increase is observed in YCSB-A, where it rises from 2.8 times the working set size to 74.2 times. This increase in data amplification is due to the coarse cache block size acting like a prefetch, bringing in more data during a local cache miss. When the workload access pattern presents less spatial locality, it results in data amplification. The data amplification only increases from 2.1 times to 2.4 times in Page Rank. The difference between workloads is due to the differing spatial locality of their access patterns. On the contrary, the AMAT decreases by an average of 56.3% with the increase in cache block size for most workloads. This is because fewer remote data fetching operations are triggered with a coarser cache block size. However, in contrast to other workloads, the AMAT increases with a coarser cache block size in YCSB-A. The reason is that the workload presents less spatial locality, and the coarser cache block size prefetches unused data which wastes the cache space.

4.6 Host Memory Capacity

We demonstrate the benefit of host memory capacity for the performance of UniMem in the scenario with limited local memory capacity (10% of working set size).

We evaluate the AMAT of UniMem by adjusting the host memory size from 0% to 80% of local memory. The host

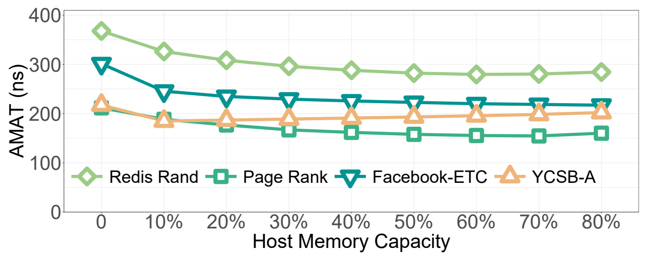
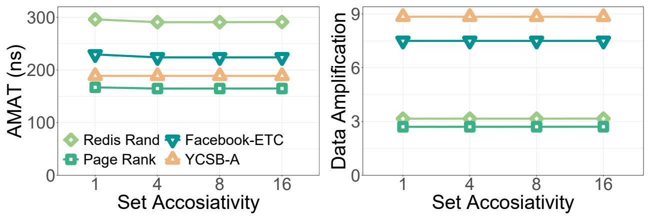


Figure 13: Impact of host memory capacity.



(a) AMAT. (b) Data Amplification.

Figure 14: Impact of set associativity.

memory is organized by OS as page cache in Linux [17]. The rest of local memory is considered as device-attached memory used for Filter-Cache. Figure 13 shows that it improves the AMAT of all workloads by 13.9% on average, as the host memory capacity increases from 0% to 10%. However, the rate of improvement slows down as the host memory capacity continues to increase. The reason is that more sub-pages are combined into 4 KB pages and promoted to host memory as the host memory expands. For workloads with less spatial locality, the page cache in host memory introduces data amplification and consumes the precious local memory.

4.7 Set Associativity

We demonstrate the influence of local cache set associativity (the count of active and inactive list pairs) on UniMem. It is important to partition data for parallel cache operations.

We evaluate both data amplification and AMAT of UniMem in the scenario with limited local cache capacity (10% of working set size) with set associativity varying from 1 to 16. Figure 14 shows that both the data amplification and AMAT of various workloads remain stable despite changes in set associativity. The overall performance of UniMem is not significantly affected by the set associativity. Thus, UniMem can fit in a range of accelerators with varying parallelization capabilities to fully leverage the hardware resources.

5 Related Work

Exposing network-attached remote memory: Advanced network technology (e.g., RDMA) provides the opportunity to expand memory capacity by memory disaggregation. Object-based [34, 35, 41, 49, 50, 54] and page-based [24, 37, 38, 44] systems treat the network-attached remote memory as a swap device. Prior cache-coherent-based DM systems [30, 31, 52] implement memory semantics (i.e., `load` and `store`) for network-attached remote memory with the help of cache-coherent interconnect. It exposes remote memory transparently over an indirection layer to isolate CNs from conflict. UniMem moves further to build a unified local-remote memory space in each CN for exposing remote memory into the physical memory space of the host, eliminating the extra software overhead introduced by the indirection layer.

Software-controlled cache: Existing DM systems [24, 30, 31, 34, 35, 37, 41, 44, 49, 50, 54] usually employ local memory to serve as local cache to absorb remote data access. They organize local data cache using different approaches, such as user-level data cache in object-based systems [34, 35, 41, 49, 50, 54], OS page cache in page-based systems [17], and 4-way associative set cache in cache-coherent-based systems [30, 31]. UniMem gives attention to design local cache resistant to thrashing and pollution. Besides, existing works neglect the perception of popular data. As a comparison, UniMem extends the cache logically using refault queue to enlarge the observation window of popular data.

Page placement strategy: In cache-coherent-based DM systems, the local memory inside CNs includes host memory and device-attached memory. The device-attached memory presents higher latency due to the limited interconnect bandwidth and interconnect overhead. Prior studies solely focus on realizing local cache on host memory [24, 37, 44] or device-attached memory [30, 31]. In comparison, UniMem proposes to utilize both types of memory by migrating popular pages from device-attached memory to host memory. Existing page migration mechanisms mainly pay attention to the page hotness [23, 25, 53, 60], while UniMem considers both page hotness and hotness fragmentation.

6 Conclusion

In this paper, we introduce UniMem that redesigns cache-coherent-based DM system with a unified local-remote memory hierarchy. UniMem carefully utilizes the host memory and device-attached memory to construct a tiered local cache for remote memory. Experiments show that UniMem is an effective solution that brings significant performance improvements across a variety of workloads. The source code of UniMem will be released in the final version of this paper.

Acknowledgement

We thank our anonymous shepherd and reviewers for their valuable feedback and suggestions. This work was supported by the National Key R&D Program of China (No. 2021YFF0704001), the Major Research Plan of the National Natural Science Foundation of China (No. 92373114), the Natural Science Foundation of China (No. 62072381), and the Natural Science Foundation of Fujian Province of China (No. 2023J06001).

References

[1] Arizona State University Twitter Data Set. https://archive.org/details/asu_twitter_dataset.

[2] Better active/inactive list balancing. https://lwn.net/Articles/495543/.

[3] CCIX. https://www.ccixconsortium.com.

[4] CXL Consortium. Compute Express Link Specifi cation Revision 3.0. https://www.computeexpresslink.org/download-the-specification.

[5] CXL Memory Challenges. https://hc34.hotchips.org/assets/program/tutorials/CXL/Hot%20Chips%202022%20CXL%20MemoryChallenges.pdf.

[6] Intel Agilex FPGA Portfolio. https://www.intel.com/content/www/us/en/products/details/fpga/agilex.html.

[7] Intel: Stratix 10 MX FPGAs. https://www.intel.com/content/www/us/en/products/details/fpga/stratix/10/mx.html.

[8] Intel Xeon Silver 4314 Processor. https://ark.intel.com/content/www/us/en/ark/products/215269/intel-xeon-silver-4314-processor-24m-cache-2-40-ghz.html.

[9] Linux kernel profiling with perf. https://perf.wiki.kernel.org/index.php/Tutorial.

[10] Memcached - A distributed memory object caching system. http://memcached.org.

[11] Memory Hot(Un)Plug. https://www.kernel.org/doc/html/latest/admin-guide/mm/memory-hotplug.html.

[12] Memtier: a command line utility for load generation and bechmarking NoSQL key-value databases. https://github.com/RedisLabs/memtier_benchmark.

[13] mm: page migration enhancement for thp. https://prod.lwn.net/Articles/726993/.

[14] Mutilate: high-performance memcached load generator. https://github.com/leverich/mutilate.

[15] Pin - A Dynamic Binary Instrumentation Tool. https://www.intel.com/content/www/us/en/developer/articles/tool/pin-a-dynamic-binary-instrumentation-tool.html.

[16] Redis: open-source, in-memory data structure store. https://redis.io/.

[17] The multi-generational LRU. https://lwn.net/Articles/851184/.

[18] TLB flush optimization. https://lwn.net/Articles/684934/.

[19] Understanding the new control groups API. https://lwn.net/Articles/679786/.

[20] VIDIA BlueField-3 data processing unit. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-3-dpu.pdf.

[21] Xilinx: Alveo U250 data center accelerator card. https://www.xilinx.com/products/boards-and-kits/alveo/u250.html.

[22] Ahmed Abulila, Vikram Sharma Mailthody, Zaid Qureshi, Jian Huang, Nam Sung Kim, Jinjun Xiong, and Wen-mei Hwu. Flatflash: Exploiting the byte-accessibility of ssds within a unified memory-storage hierarchy. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 971–985, 2019.

[23] Neha Agarwal and Thomas F Wenisch. Thermostat: Application-transparent page management for two-tiered main memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 631–644, 2017.

[24] Marcos K Aguilera, Nadav Amit, Irina Calciu, Xavier Deguillard, Jayneel Gandhi, Stanko Novakovic, Arun Ramanathan, Pratap Subrahmanyam, Lalith Suresh, Kiran Tati, et al. Remote regions: a simple abstraction for remote memory. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 775–787, 2018.

[25] Hasan Al Maruf and Mosharaf Chowdhury. Effectively prefetching remote memory with leap. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 843–857, 2020.

[26] Emmanuel Amaro, Christopher Branner-Augmon, Zhihong Luo, Amy Ousterhout, Marcos K Aguilera, Aurojit Panda, Sylvia Ratnasamy, and Scott Shenker. Can far memory improve job throughput? In *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys 20)*, pages 1–16, 2020.

[27] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. Workload analysis of a large-scale key-value store. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems*, pages 53–64, 2012.

[28] Shai Bergman, Priyank Faldu, Boris Grot, Lluís Vilanova, and Mark Silberstein. Reconsidering os memory optimizations in the presence of disaggregated memory. In *Proceedings of the 2022 ACM SIGPLAN International Symposium on Memory Management*, pages 1–14, 2022.

[29] Ankit Bhardwaj, Todd Thornley, Vinita Pawar, Reto Achermann, Gerd Zellweger, and Ryan Stutsman. Cache-coherent accelerators for persistent memory crash consistency. In *Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems*, pages 37–44, 2022.

[30] Irina Calciu, M Talha Imran, Ivan Puddu, Sanidhya Kashyap, Hasan Al Maruf, Onur Mutlu, and Aasheesh Kolli. Rethinking software runtimes for disaggregated memory. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 79–92, 2021.

[31] Irina Calciu, Ivan Puddu, Aasheesh Kolli, Andreas Nowatzyk, Jayneel Gandhi, Onur Mutlu, and Pratap Subrahmanyam. Project pberry: Fpga acceleration for remote memory. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, pages 127–135, 2019.

[32] Asaf Cidon, Assaf Eisenman, Mohammad Alizadeh, and Sachin Katti. Dynacache: Dynamic cloud caching. In *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, 2015.

[33] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154, 2010.

[34] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. {FaRM}: Fast remote memory. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 401–414, 2014.

[35] Aleksandar Dragojević, Dushyanth Narayanan, Edmund B Nightingale, Matthew Renzelmann, Alex Shamis, Anirudh Badam, and Miguel Castro. No compromises: distributed transactions with consistency, availability, and performance. In *Proceedings of the 25th symposium on operating systems principles*, pages 54–70, 2015.

[36] Donghyun Gouk, Sangwon Lee, Miryeong Kwon, and Myoungsoo Jung. Direct access,{High-Performance} memory disaggregation with {DirectCXL}. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 287–294, 2022.

[37] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G Shin. Efficient memory disaggregation with infiniswap. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 649–667, 2017.

[38] Zhiyuan Guo, Yizhou Shan, Xuhao Luo, Yutong Huang, and Yiying Zhang. Clio: A hardware-software codesigned disaggregated memory system. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 417–433, 2022.

[39] Jian Huang, Anirudh Badam, Moinuddin K Qureshi, and Karsten Schwan. Unified address translation for memory-mapped ssds with flashmap. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, pages 580–591, 2015.

[40] Frans Kaashoek, Robert Morris, and Yandong Mao. Optimizing mapreduce for multicore architectures. 2010.

[41] Anuj Kalia, Michael Kaminsky, and David G Andersen. Using rdma efficiently for key-value services. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, pages 295–306, 2014.

[42] Anuj Kalia, Michael Kaminsky, and David G Andersen. Design guidelines for high performance rdma systems. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, pages 437–450, 2016.

[43] Huaicheng Li, Daniel S Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, et al. Pond: Cxl-based memory pooling systems for cloud platforms. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 574–587, 2023.

[44] Shuang Liang, Ranjit Noronha, and Dhabaleswar K Panda. Swapping to remote memory over infiniband: An approach using a high performance network block device. In *2005 IEEE International Conference on Cluster Computing*, pages 1–10. IEEE, 2005.

[45] Ling Liu, Wenqi Cao, Semih Sahin, Qi Zhang, Juhyun Bae, and Yanzhao Wu. Memory disaggregation: Research problems and opportunities. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 1664–1673. IEEE, 2019.

[46] Yucheng Low, Joseph E Gonzalez, Aapo Kyrola, Danny Bickson, Carlos E Guestrin, and Joseph Hellerstein. Graphlab: A new framework for parallel machine learning. *arXiv preprint arXiv:1408.2041*, 2014.

[47] Teng Ma, Kang Chen, Shaonan Ma, Zhuo Song, and Yongwei Wu. Thinking more about rdma memory semantics. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 456–467. IEEE, 2021.

[48] Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit Kanaujia, and Prakash Chauhan. Tpp: Transparent page placement for cxl-enabled tiered-memory. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 742–755, 2023.

[49] Christopher Mitchell, Yifeng Geng, and Jinyang Li. Using {One-Sided}{RDMA} reads to build a fast,{CPU-Efficient}{Key-Value} store. In *2013 USENIX Annual Technical Conference (USENIX ATC 13)*, pages 103–114, 2013.

[50] Jacob Nelson, Brandon Holt, Brandon Myers, Preston Briggs, Luis Ceze, Simon Kahan, and Mark Oskin. {Latency-Tolerant} software distributed shared memory. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 291–305, 2015.

[51] Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C Li, Ryan McElroy, Mike Paleczny, Daniel Peek, Paul Saab, et al. Scaling memcache at facebook. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 385–398, 2013.

[52] Christian Pinto, Dimitris Syrivelis, Michele Gazzetti, Panos Koutsovasilis, Andrea Reale, Kostas Katrinis, and H Peter Hofstee. Thymesisflow: A software-defined, hw/sw co-designed interconnect stack for rack-scale memory disaggregation. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 868–880. IEEE, 2020.

[53] Amanda Raybuck, Tim Stamler, Wei Zhang, Mattan Erez, and Simon Peter. Hemem: Scalable tiered memory management for big data applications and real nvm. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 392–407, 2021.

[54] Zhenyuan Ruan, Malte Schwarzkopf, Marcos K Aguilera, and Adam Belay. {AIFM}:{High-Performance},{Application-Integrated} far memory. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 315–332, 2020.

[55] Vivek Seshadri, Onur Mutlu, Michael A Kozuch, and Todd C Mowry. The evicted-address filter: A unified mechanism to address both cache pollution and thrashing. In *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, pages 355–366, 2012.

[56] Debendra Das Sharma. Compute express link®: An open industry-standard interconnect enabling heterogeneous data-centric computing. In *2022 IEEE Symposium on High-Performance Interconnects (HOTI)*, pages 5–12. IEEE, 2022.

[57] Jiaxin Shi, Youyang Yao, Rong Chen, Haibo Chen, and Feifei Li. Fast and concurrent {RDF} queries with {RDMA-Based} distributed graph exploration. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 317–332, 2016.

[58] Shin-Yeh Tsai, Yizhou Shan, and Yiying Zhang. Disaggregating persistent memory and controlling them remotely: An exploration of passive disaggregated {Key-Value} stores. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 33–48, 2020.

[59] Qing Wang, Youyou Lu, and Jiwu Shu. Sherman: A write-optimized distributed b+ tree index on disaggregated memory. In *Proceedings of the 2022 International Conference on Management of Data*, pages 1033–1048, 2022.

[60] Zi Yan, Daniel Lustig, David Nellans, and Abhishek Bhattacharjee. Nimble page management for tiered memory systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 331–345, 2019.

[61] Juncheng Yang, Yazhuo Zhang, Ziyue Qiu, Yao Yue, and Rashmi Vinayak. Fifo queues are all you need for cache eviction. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 130–149, 2023.

[62] Yang Yang, Sanmukh R Kuppannagari, Ajitesh Srivastava, Rajgopal Kannan, and Viktor K Prasanna. Fasthash: Fpga-based high throughput parallel hash table. In *High Performance Computing: 35th International Conference, ISC High Performance 2020, Frankfurt/Main, Germany, June 22–25, 2020, Proceedings 35*, pages 3–22. Springer, 2020.

[63] Ruizhi Zhang, Sasindu Wijeratne, Yang Yang, Sanmukh R Kuppannagari, and Viktor K Prasanna. A high throughput parallel hash table on fpga using xor-based memory. In *2020 IEEE High performance extreme computing conference (HPEC)*, pages 1–7. IEEE, 2020.